%ARRAY: construction and usage of arrays of macro variables

Ronald Fehd, Centers for Disease Control, and Prevention, Atlanta GA

ABSTRACT

The SAS® software data step statement array V {3} $ V1-V3 ('A' 'B' 'C'); produces three character variables named V1, V2, and V3 with corresponding initial values, 'A', 'B', and 'C' and a function, dim(V), which returns a value of 3. Programmers can write simple macro tools for use in larger macro procedures. These tools can duplicate SAS software data step constructions that the programmer is comfortable using and make reading and comprehension easier. The macro statement %ARRAY(V,A B C) produces three macro variables, V1, V2, and V3, with corresponding values: A, B, and C and macro variable DIM_V with the value 3. These variables can then be used in the macro iterative loop statement %DO I = 1 %TO &DIM_V.;

This paper examines the SAS data step array statement and discusses the issues in constructing and using arrays of macro-variables. The macro ARRAY takes parameters of either a list of elements or a data set.

Macro ARRAY is a basic utility used in two other macros that address analysis of multiple-response data. See Fehd (1996), (1997) %CHECKALL and %SHOWCOMB.

INTRODUCTION

A common task for an experienced programmer is to recognize a recurring pattern of code and encapsulate that pattern in a routine which simplifies the processing presentation, while still enabling later readers of the program to grasp the complex concepts that have been coded.

The SAS software macro language is a simple yet powerful programming language. This article examines the SAS software array and associated do loop statements with the idea of translating those concepts into SAS software macro language usage.

SAS array statement & dimension (dim) function

The explicit array statement in SAS software has seven phrases; we will examine the four that are most commonly used:
1. ARRAY is the SAS statement key-word
2. array-name: required
3. subscript: required, either of
   {3}: number supplied for creation of a series of variables
   {*}: asterisk indicating subscript is determined by SAS software by counting the supplied array-elements
4. array-elements: an optional list of variable names

The dimension function has two commonly used phrases:
1. DIM is the SAS function name
2. parameter is an array-name defined in same data step

A typical usage of the array statement would consist of accessing one set of variables in order to repeat some processing on each variable. The example in program 1 below reads in three Fahrenheit temperatures and converts them to Celsius. Note that the proc CONTENTS listing shows that SAS has created a series of variables based on the absence of the array-elements in the array Celsius statement. Their names -- Celsius1, Celsius2, and Celsius3 -- correspond to the way the variables are accessed in the iterative loop by the array convention of Celsius{1}, Celsius{2}, and Celsius{3}.

```
Program 1
data TEMPRATR;
 input Low Med Hi;
 array Celsius {3};*note no array-elements, see CONTENTS;
 array Farnheit {*} Low Med Hi;
 do I = 1 to dim(Farnheit);
  Celsius{I} = (Farnheit{I} - 32) * 5/9;      end;
cards;*<deletion>;
proc CONTENTS;

- - - SAS output: - - -
#    Variable    Type    Len    Pos
-    --------    ----    ---    ---
4    CELSIUS1    Num       8     24
5    CELSIUS2    Num       8     32
6    CELSIUS3    Num       8     40
3    HI          Num       8     16
1    LOW         Num       8      0
2    MED         Num       8      8
```

SAS software macro language iterative loop

To replicate the SAS software iterative loop in the macro language we use a sequentially numbered series of macro variables and a macro variable containing the dimension:

```
%LET VAR1 = Q04A;
%LET VAR2 = Q04B;
%LET VAR3 = Q04C;
%LET DIM_VAR = 3;
```

The macro iterative loop and usage of the macro variables can then be written in a form that is visually similar to the SAS software iterative loop.

```
%DO I = 1 %TO &DIM_VAR;
   %PUT VAR&I. :: &&VAR&I.;   %END;
```

This loop writes the following note to the SAS log:

```
VAR1 :: Q04A
VAR2 :: Q04B
```

```
VAR3 :: Q04C
```

This is a construction used regularly in certain types of macros. The purpose of this paper is to construct a macro that supports this iterative loop. Such a macro would be named ARRAY, and would have two of the SAS array statement phases as parameters: array-name, and array-element values. This macro would return a sequentially-numbered series of macro variables and the dimension of the array. The array-element values could be either a provided list or the values of a variable in a data set. This second option of providing the array-element values in a data set would enable macro procedures to be completely data-driven. See Fehd (1996) %CHECKALL and %SHOWCOMB for examples.

Parameters and Constraints

The simplicity of the macro language both allows and requires construction of a routine that has the appearance of the SAS software array statement. Since this is a routine and not a SAS software implementation, there are relations among the parameters that are constraints.

The first and most obvious is that the array-name parameter must follow SAS naming conventions. SAS names may be up to eight characters in length. For this routine, some number of characters must be reserved for the sequential numbering of the suffix. As the magnitude of the number of array-elements increases, the length of the array-name must decrease in order for the combined length to be less than or equal to eight.

A second constraint on the array-name parameter is that the macro variable used for the dimension has the form: DIM_<array-name>. This construction was chosen to appear visually similar to the usage of the dimension function: dim(<array-name>). This convention reduces the length of the array-name as prefix to four characters.

The array-name parameter is both prefix and suffix. As suffix to the name of the returned value of dimension, it can be no more than four characters in length. As prefix to the series of macro variables four characters in the array-name allows a maximum of 9,999 sequentially numbered macro variables to be created without suffering a ‘SAS name too long’ error. For larger arrays, the length of the array-name can be as small as one character.

Array-elements in the SAS software data step array statement are assumed to be delimited by spaces. When array-element values are provided to this routine as a list, the macro scan function is used to pick out each value. The delimiters of the macro function %scan are the set of non-alpha-numeric characters. For special cases where, for instance, an array-element value may contain two or more words, the delimiter parameter may be supplied.

A data set and variable name may be supplied as parameters, instead of a list. This routine was written to handle various series of variable names, which were subsets of a proc CONTENTS output data set. Review the test data with the macro.

Case 1: Scanning macro values from a list

The macro function scan operates the same as the SAS software function. In order to construct a loop which has a data-dependent termination, it is necessary to use and test a temporary variable for the exit condition. Here is pseudo-code for a loop that converts a list to array-elements:

```
initialize: I := 1
            pick I-th ITEM from ITEMLIST
loop:       assign ITEM to macro-variable
            increment I
            pick I-th ITEM from ITEMLIST
until (ITEM is blank)
```

Whereas the pseudo-code shows that the test is done at the bottom of the loop, SAS attaches the until function to the iterative %DO at the top of the loop. As noted in the Caution section below, the macro variables are global. The index is incremented using the %eval function. At the loop exit the index is off by one; the dimension is therefore index - 1.

Case 2: Symput: macro values from a data set variable

SAS software provides the symput function to transfer values from a data set variable to the macro environment. The symput function takes two arguments, macro-variable name, and macro-variable value.

```
symput(mac-var name, mac-var value)
```

The macro-variable name is a character expression consisting of the array-name prefix plus a suffix which is the series of integers from one to the number of observations of the data set. The macro-variable value is the value of the data set variable.

```
symput(prefix + suffix, variable name)
```

The prefix is a macro variable and is to be evaluated as a quoted string. Double exclamation marks -- !! -- are used as character-value concatenation operator. The suffix is an integer -- here, the SAS observation counter -- converted to a character expression.

```
symput("&ARRAY-NAME" !! left(_N_), variable name)
```

Usage of %ARRAY in other macros

The code for creating a macro array from a list was first written as part of the %CHECKALL macro. This macro analyzes multiple-response data, a series of variables which contain answers to survey questions with the instructions ‘check all that apply’. After typing in hundreds of variables as lists for the various series, I wrote the second section which uses a previously prepared subset of a proc CONTENTS data set. This addition allows both research and production usage of the %CHECKALL macro. See Fehd, 1996, %CHECKALL, and %SHOWCOMB and test data with the macro.

DiIorio, 1996, discusses macro arrays of data set names.

Caution

SAS provides no method to limit the scope of macro variables. The cost of hiding the details of converting a list to a series of macro variables is that the macro variables returned are global variables.

CONCLUSION

The SAS software array and do statements are a simple programming tool which allow a programmer to access a list of variables. The macro language allows a programmer to access a list of items with a %DO statement but lacks a specific %ARRAY statement. This paper has presented a macro ARRAY which converts either a list or values of a variable into a sequentially-numbered series of macro-variables with common prefix and sequential numeric suffix and also returns a macro-variable with the dimension. This routine hides complexity and simplifies readability of programs which contain macro loops.

The SAS software macro language is a simple language. It's simplicity leaves many advanced programming concepts apparently unavailable. It's simplicity is an asset in that, with some forethought and planning, generic tools can be relatively easily written. This macro was initially developed to take a list of variable names as a parameter. After some usage it became apparent that adding the option to accept a data set as parameter would eliminate tedious typing of the variable lists, and, in addition, since the routine was then data-driven, guarantee the accuracy of the data thus processed.

REFERENCES

DiIorio, Frank (1996), "MACARRAY: a Tool to Store Dataset Names in a Macro 'Array'." Proceedings of the Fourth Annual Conference of the SouthEast SAS Users Group, 229-231.

Fehd, Ronald (1996)," %CHECKALL, a macro to produce a frequency of response data set from multiple-response data." Proceedings of the Fourth Annual Conference of the SouthEast SAS Users Group, 393-398.

Fehd, Ronald (1996), "%SHOWCOMB: a macro to produce a data set with frequency of combinations of responses from multiple-response data." Proceedings of the Fourth Annual Conference of the SouthEast SAS Users Group, 399-402.

Fehd, Ronald (1997) %ARRAY, %CHECKALL, %SHOWCOMB: Proceedings of the Twenty-Second Annual SAS Users Group International Conference.

Author: Ronald Fehd
Centers for Disease Control
4770 Buford Hwy NE   MS-G25
Atlanta GA 30341-3724   voice: 770/488-4316
e-mail: RJF2@phpdLs1.em.cdc.gov   (D eL S one)
SAS-L archives: send e-mail
to: SAScontrib@SASserv.uga.edu
for %ARRAY   subject: cntb0031: download
for %CHECKALL   subject: cntb0032: download
for %SHOWCOMB subject: cntb0033: download

```
 /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*  MACRO:  ARRAY                                                         *
*  PARAMETERS:                                                           *
*    array-name                                                          *
*    array-element values: either of                                     *
*      horizontal list of array element values                           *
*                      delimiters may be specified, see default list     *
*      vertical   list: data set w/ variable containing array values     *
*  USAGE: 1. %_ARRAY(ARAYNAME,ITEM-LIST);                                *
*         2. %_ARRAY(ARAYNAME,ITEM-LIST,DELIMITR=/);                     *
*         3. %_ARRAY(ARAYNAME,DATA=DATANAME,VAR=Var-Name);               *
*  DESCRIPTION:                                                          *
*    This macro returns a series of global macro-variables               *
*    named &ARAYNAME.1  &ARAYNAME.2 .. &ARAYNAME.n                       *
*    and a macro-variable named DIM_&ARAYNAME.                           *
*    i.e. %ARRAY(VAR,Q04A Q04B Q04C);                                    *
*    returns:    VAR1::Q04A VAR2::Q04B VAR3::Q04C, DIM_VAR::3            *
*  NOTES:                                                                *
*    length(ARAYNAME) must be <= 4                                       *
*    length(ARAYNAME) + length(dim_ARAYNAME) must be <= 8                *
*  CAUTION: creates %GLOBAL macro-variables                              *
*  PROCESS: case 1: scan user-provided list of array-elements            *
*           case 2: data set: call symput of variable                    *
*           case 3: error msg                                            *
*  KEYWORDS: array %ARRAY CONTENTS delimiter loop macro PARMBUFF %scan   *
*            symput SYSPBUFF %until                                     */
%MACRO ARRAY(/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
 ARAYNAME    /*macro-var array-name: len(array-name) must be <= 4       */
,ITEMLIST    /*horizontal list of array-element values                  */
,DATA    =.  /*vertical list: data set name                             */
,VAR     =.  /*vertical list: variable in data set                      */
,DELIMITR=%STR( ).<%str(()]&!$*%str())^-/%>\/*delimiter of horiz-list */
)/PARMBUFF;/* see SYSPBUFF usage in error msg -------------------------*/
%IF  "&ARAYNAME" ne ""/*- case 1: ARAYNAME & ITEMLIST -----------------*/
 and "&ITEMLIST" ne "" and %length(&ARAYNAME) le 4 %THEN %DO;
 %local I ITEM;
```

```
 %LET I    = 1;
 %LET ITEM = %qscan(&ITEMLIST,&I.,&DELIMITR);
 %DO %until(&ITEM = ); %global &ARAYNAME.&I;
                      %LET &ARAYNAME.&I = &ITEM.;
                      %LET    I       = %eval(&I + 1);
                      %LET    ITEM    = %qscan(&ITEMLIST,&I.,&DELIMITR);
                                                  /* %DO %until*/ %END;
 %global DIM_&ARAYNAME;
 %LET    DIM_&ARAYNAME = %eval(&I - 1);
 %PUT macro proc ARRAY returns array: &ARAYNAME, dim=&&DIM_&ARAYNAME.;
 %PUT from list <&ITEMLIST.>;/*............ ARAYNAME & ITEMLIST */ %END;
%ELSE %IF "&ARAYNAME" ne "" /*- case 2: ARAYNAME & (DATA & VAR) ------*/
      and "&ITEMLIST" eq "" and "&DATA" ne "." and "&VAR" ne "."
      and %length(&ARAYNAME) le 4 %THEN %DO;
proc CONTENTS data = &DATA   (keep = &VAR.) noprint
              out  = CONTENTS(keep = Nobs Type);
%global DIM_&ARAYNAME;
DATA _NULL_;
 length VarType $ 1;
 set CONTENTS;
 if Type = 1 then VarType = 'N';
 else             VarType = 'C';
 call symput("VARTYPE" ,VarType);
 call symput("DIM_&ARAYNAME",trim(left(put(Nobs,32.))));
stop; run;
%local I;
%DO    I = 1 %TO &&DIM_&ARAYNAME.; %global &ARAYNAME.&I.;          %END;
DATA _NULL_;
 retain I 0;
 do until(EndoFile);
   set &DATA. end = EndoFile;
   I + 1;/* symput(mac-var name : prefix + suffix,
                  mac-var value: variable name */
   call symput("&ARAYNAME" !!        left(put(I   ,     7.0 ) ),
                                 trim(left(
     %IF "&VARTYPE" = "N" %THEN           put(&VAR,best32.16)          ;
     %ELSE                                       &VAR                        ;
                                                  /*symput closure*/ )));
                                              /*do until(EndoFile)*/ end;
stop; run;
%PUT macro proc ARRAY returns array: &ARAYNAME, dim=&&DIM_&ARAYNAME.;
%PUT from data = &DATA, var = &VAR.;/*...ARAYNAME & (DATA & VAR)*/ %END;
%ELSE %DO;/* print error msg ----------------------------------------*/
  %PUT ERROR: in macro ARRAY:;
  %PUT array-name : required;
  %PUT array-elements may be list, or (data and var);
  %PUT list     : ARRAY(ARAYNAME,ITEM-LIST);
  %PUT data+var: ARRAY(ARAYNAME,DATA=DATA-NAME,VAR=VAR-NAME);
  %IF %length(&ARAYNAME) gt 4 %THEN
   %PUT length(ARAYNAME) must be <= 4;
  %PUT parmlist: <&SYSPBUFF.>;/*from /PARMBUFF macro statement option*/
  %PUT parm: ARAYNAME = <&ARAYNAME.> length = <%length(&ARAYNAME)>;
  %PUT parm: ITEMLIST = <&ITEMLIST.>;
  %PUT parm: DATA     = <&DATA.>    ;
  %PUT parm: VAR      = <&VAR.>     ;/*.............. ANY ERRORS*/ %END;
```

```
 /*..................................................... ARRAY */ %MEND;
 /*-test data ------------ enable by ending this line with slash (/) **
DATA VQ04;  *convention for %CHECKALL setup;
length Name $ 8;  *can use proc CONTENTS output data set;
Name = ‘Q04A’; output;
Name = ‘Q04B’; output;
Name = ‘Q04C’; output; run;
%MACRO DEMO(DATA_OUT,ITEMLIST);* typical usage ----------------------*;
%IF &ITEMLIST ne %THEN %ARRAY(VAR, &ITEMLIST.);
%ELSE %ARRAY(VAR, DATA = V&DATA_OUT., VAR = Name);
%DO I = 1 %TO &DIM_VAR.; %PUT VAR&I. :: &&VAR&I.;                   %END;
 *...................................................... DEMO *; %MEND;
%DEMO(Q03, Q03A Q03B Q03C);  *data exploration usage;
%DEMO(Q04);  *production usage;
%ARRAY(TOOBIG,A B C D E);*see error msg;
%PUT _user_;*show list of macro-variables;
RUN; /*................................................ END TEST DATA*/
```